

Peter Rogers

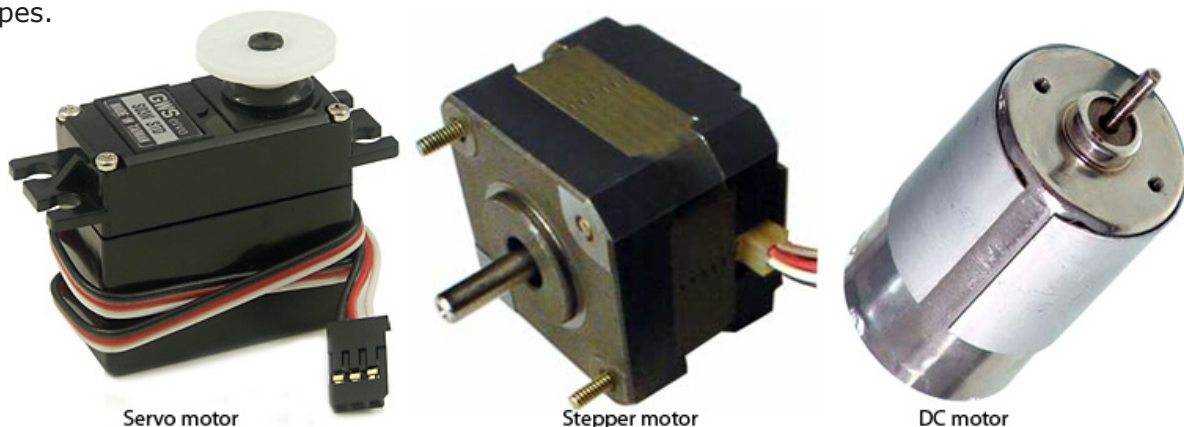
# Physical Computing workshops

## Making movement

This session will be focussed on making movement with motors and other actuators. This is the fundamental step in beginning to make things 'outside the computer' and gives you opportunity to be truly physical, enabling you to create work that can physically act with and respond to its environment.

This session will give you an introduction to controlling DC motors, stepper motors and servo motors. You will also be introduced pneumatics and linear solenoids. From these building blocks you can make all types of motion and control possible.

As part of this session you will also have a short intro to making small sensor circuits from items such as buttons and light sensors. The circuit used can be reused for many other sensor types.



## Motors

Motors come in three main flavours, Servo, DC and Stepper. They all make rotational movement but after this they differ.

**DC** motors are the simplest motor, they turn continuously at speeds dependent on the amount of voltage going to the motor. DC motors can turn both ways and this can be controlled by setting one wire to high and one low. You can also control the speed of a DC motor through PWM. DC motors do not have any idea of 'where they are' so they are difficult to use to position things accurately on its own.

**Servo** motors are used to make accurate positional movement. They only have a limited rotation of around 270 degrees so can't generally be used for continuous rotation. Their speed and accuracy can be controlled very easily and precisely.

**Stepper** motors are used in printers, CD drives amongst many other things, so they are very accurate and they offer full rotation and high precision. They are also very strong offering a lot of torque or 'pulling power'. They are however the hardest motor to control, but there are libraries out there to make it simpler.

## Controlling a Servo

Servo motors vary, you may see on the net that you can get 360 degree servos and this is true but they will not give feedback on position, classic servo will know where it is and will go where it is told to but only from 0 to 270 degrees.

Servos are easy to control from an arduino and it is possible to control a maximum of 4 per 1 arduino which could mean a lot of movement.

The code below takes an input from a sensor and moves the servo dependent upon the input. Copy and paste the code into arduino.

```
/**
simple Servo
completed at 15.50 16/01/2009
Created by pete-rogers.co.uk
Servo mover moves a servo dependent upon a analog sensor signal from 0 - 1024
Connectors:
Servo to pin 10
Analog sensor to pin 2
**/
int servoPin = 10;
int sensorPin = 2;
int lower = 600; //max in
int higher = 2500; //max out
int val;

void setup(){
  Serial.begin(9600);
}

void loop (){
  delay(20);
  Serial.println(val);
  checkSensor();
  servoMove();
}

//checks the sensor and adjusts to servos range
int checkSensor(){
  val = analogRead(sensorPin);
  val = val * 2;
  val = val + lower;
  if(val < lower){
    val = lower;
  }
  if (val > higher){
    val = higher;
  }
}

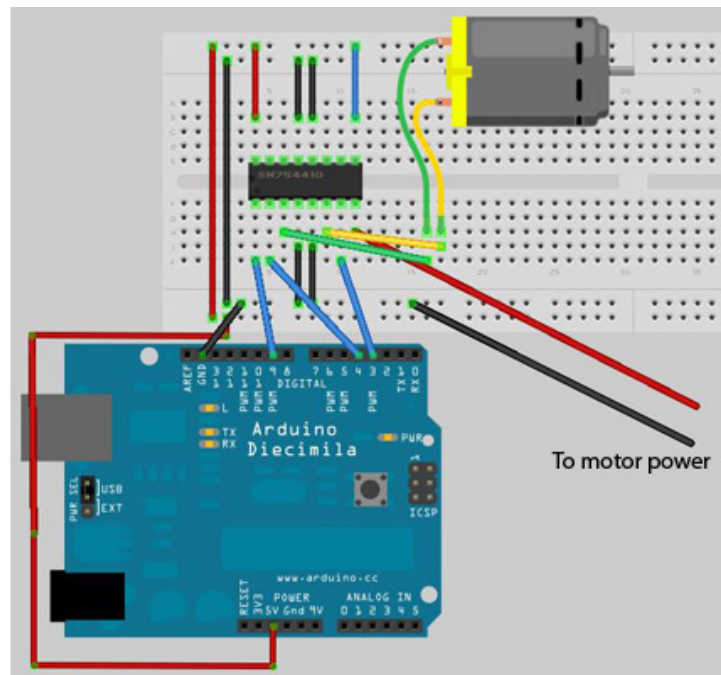
void servoMove(){
  // Turn the motor on
  digitalWrite(servoPin, HIGH);
  // Length of the pulse sets the motor position
  delayMicroseconds(val);
  // Turn the motor off
  digitalWrite(servoPin, LOW);
}
```

In the code above there are two main variables 'higher and 'lower' that need to be configured. These numbers represent the maximum and minimum rotation of the servo and might need to be changed dependent on the make of servo. If you try to make the servo go further than it can go then the servo will make a horrible noise and eventually burn out, so calibrate the servo by correcting these numbers so that the servo is not trying to move out of its extent.

The servo is positioned by sending a tiny microsecond pulse that the servo translates to the position in its range. This pulse is in microseconds; the 'lower' variable is 600 microseconds, when it is sent to the servo the servo will position itself at its lowest position or 0 degrees and when it is sent 2500 microseconds it will move to 270 degrees its highest position.

## Controlling a DC motor with an H-Bridge

DC motors are very easy to control they can move in both directions, can go on rotating forever and can change speed. The only major downside is that you can't control their position of rotation exactly and without gearing they are quite weak or don't have much torque.



The above image is the breadboard scheme for attaching one DC motor to an arduino. The chip on the breadboard is called an H-Bridge which is a common chip for this type of work. You need an H-Bridge because a DC motor needs more voltage and current than the arduino can give, so you need to run the motor from another power source and the H-Bridge controls power change and acts as a barrier for the arduino so you don't suffer power surges which can blow your board.

There are three connections from the arduino to the H-Bridge on pins 3, 4 and 9.

Pins 3 and 4 control the direction of the motor so if pin 3 is HIGH and pin 4 is LOW the DC motor will rotate clockwise and if 3 is LOW and 4 is HIGH it will rotate anti-clockwise.

Pin 9 is a PWM (pulse width modulation) pin and this controls the speed of rotation from 0 = no movement 255 = highest speed.

The arduino code below controls the direction and speed of the DC motor, copy and paste into arduino. You need to attach a sensor to analog pin 0 and a button or some such input to digital pin 12 so that the motor can change direction.

```
/**
Simple DC motor controller using H-Bridge;
Created 17.00 16/01/2009 Pete Rogers
**/
int motorPin0 = 3;
int motorPin1 = 4;
int speedPin = 9;
int sensorPin = 2;
int button = 12;
int val = 0;
boolean forward = true;

void setup() {

pinMode(motorPin0, OUTPUT);
pinMode(motorPin1, OUTPUT);
pinMode(speedPin, OUTPUT);
pinMode(button, INPUT);
```

```
}  
  
void loop() {  
  if (digitalRead(button)== HIGH){  
    forward = false;  
  }else{  
    forward = true;  
  }  
  if (forward == true){  
    digitalWrite(motorPin0, LOW);  
    digitalWrite(motorPin1, HIGH);  
  }else{  
    digitalWrite(motorPin0, HIGH);  
    digitalWrite(motorPin1, LOW);  
  }  
  analogRead(sensorPin);  
  analogWrite(speedPin, val);  
}
```